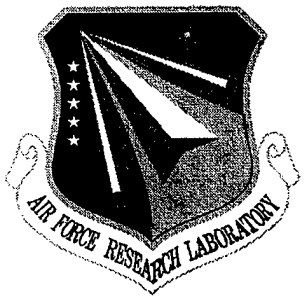
AFRL-SN-RS-TR-1999-233
Final Technical Report
October 1999



ADVANCED SIGNAL PROCESSING ALGORITHM EVALUATION

Integrated Sensors, Inc.

Milissa Benincasa and Dave Nobles

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-SN-RS-TR-1999-233 has been reviewed and is approved for publication.

APPROVED: Russell D. Brown

Russell D. Brown
Project Engineer

FOR THE DIRECTOR: Robert G. Polce

Robert G. Polce, Acting Chief

Rome Operations Office

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/SNRT, 26 Electronic Pky, Rome, NY 13441-4514. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	OCTOBER 1999	Dec 94 - Jan 99

4. TITLE AND SUBTITLE
ADVANCED SIGNAL PROCESSING ALGORITHM EVALUATION

5. FUNDING NUMBERS
C - F30602-94-C-0027
PE - 63762E
PR - 4506
TA - SN
WU - IT

6. AUTHOR(S)
Milissa Bennincasa and Dave Nobles

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Integrated Sensors, Inc.
502 Court Street Suite 210
Utica NY 13502

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/SNRT
26 Electronic Pky
Rome NY 13441-4514

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-SN-RS-TR-1999-233

11. SUPPLEMENTARY NOTES
Air Force Research Laboratory Project Engineer: Russell D. Brown/SNRT/(315) 330-2661

12a. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The primary objective of this program was to develop software for evaluation of radar signal processing algorithms. Real time processing of radar data generally requires parallel computer architectures that present challenging programming problems. Under this effort, tools were developed to streamline the transition from concept to processing software required to operate in real time. These tools included a Multisense Algorithm Development System (MADE), a parallel processing architecture manager, a configurable interface for AFRL Surveillance Laboratory hardware and software interconnections, and a development environment for building applications and tracking performance. Experiments were performed in the laboratory to demonstrate radar signal processing and target detection in real time.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Radar Signal Processing, Sensor Capability, Parallel Processing, Rapid Prototyping	32
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Standard Form 298 (Rev. 2-89) (EG)
Prescribed by ANSI Std. 239.18
Designed using Perform Pro, WHS/DIOR, Oct 94

TABLE OF CONTENTS

1 Program Summary and Results....1
1.1 Background....1
1.2 Program Objectives....1
1.3 Summary of Program Accomplishment....1

2 Advanced Signal Processing Algorithm Evaluation Program....2
2.1 Concept Overview....2
2.2 MADS Environment....6
2.3 Radar Signal Processing Application Developed Using MADS....9

3 Evolution of MADS....10
3.1 Real-Time Development Environment (RTDE) Concept....10
3.2 RTDE Components....12
3.3 RTDE Real-Time Performance Monitoring....14
3.4 RTDE Enhances MADS Capabilities....15

4 RTEXPRESS™ Commercialization of RTDE....17

5 Future Directions....18

6 References....19

List of Figures

Figure 1 Typical Software Development Process Cycle....2
Figure 2 Modified Software Development Process Cycle....3
Figure 3 Multisensor Algorithm Development System....4
Figure 4 Conceptual Organization of the MADS Facility....5
Figure 5 MADS Environment Components....6
Figure 6 System Configuration of Typical Experiment....7
Figure 7 Control and Data Visualization Objects....8
Figure 8 Radar Signal Processing Application....9
Figure 9 Dataflow Through MADS....10
Figure10 Real-Time Performance Monitoring....12
Figure11 Target Balancing Tool....13
Figure12 Real-Time Performance Monitoring....14
Figure13 RTEshot Post-Mortem Instrumentation....17

List of Tables

Table 1. Comparison of Program Capabilities.....15

1 Program Summary and Results

1.1 Background

The Air Force Research Laboratory Rome Site (AFRL) houses a state-of-the-art Radar Surveillance Laboratory (RSL) that can be used to evaluate and develop real-time signal processing algorithms. In order to provide better utilization of this facility the need existed to integrate the RSL hardware and software components so as to reduce the separate setup time required to interconnect the various laboratory equipment. This also required a software development tool that would significantly reduce the overall algorithm development time for evaluating and implementing the real-time radar signal and track processing on the RSL hardware.

1.2 Program Objectives

The objective of this effort was to build a Multisensor Algorithm Development System (MADS) that would integrate the RSL hardware and software. Part of the MADS concept was to provide an interactive environment for developing adaptive signal processing algorithms for low observable target detection. MADS was focused on providing the following features:

1. A configurable interface that would allow RSL hardware and software to be interconnected.
2. A software development environment for building real-time signal/track processing applications.
3. A parallel processing architecture manager that would allow the implementation of development functions onto existing i860 based processing hardware.

The MADS environment was to be demonstrated by providing a basic end-to-end radar processing algorithm set for baselining performance and daily operation.

1.3 Summary of Program Accomplishment

A major accomplishment of the Advanced Signal Processing Algorithm Evaluation (ASPAE) program was the development of the MADS software development environment. MADS has provided the foundation for a commercially available rapid prototyping environment that allows MATLAB® code to be directly executed on embedded real-time and non-embedded real-time computing platforms. MADS provided the entrée into a DARPA/ITO funded research effort that produced the current Integrated Sensors, Inc. (ISI) commercial product RTExpress™.

2 Advanced Signal Processing Algorithm Evaluation Program

2.1 Concept Overview

The overall objective of the Advanced Signal Processing Algorithm Evaluation Program was to improve, demonstrate, and evaluate adaptive signal processing algorithms for low observable target detection. The goal was to develop an efficient algorithm implementation facility to easily compare the performance of competing techniques relative to processing requirements, false alarm rate, detection capability and track formation as a function of signal to clutter ratio.

ISI's innovative approach was to introduce the concept of directly utilizing MATLAB® algorithm code on the target signal processing hardware for the implementation of these candidate applications. This approach was selected to reduce the software development cycle costs associated with developing and debugging adaptive signal processing application code. The typical software development process cycle is depicted in figure 1.

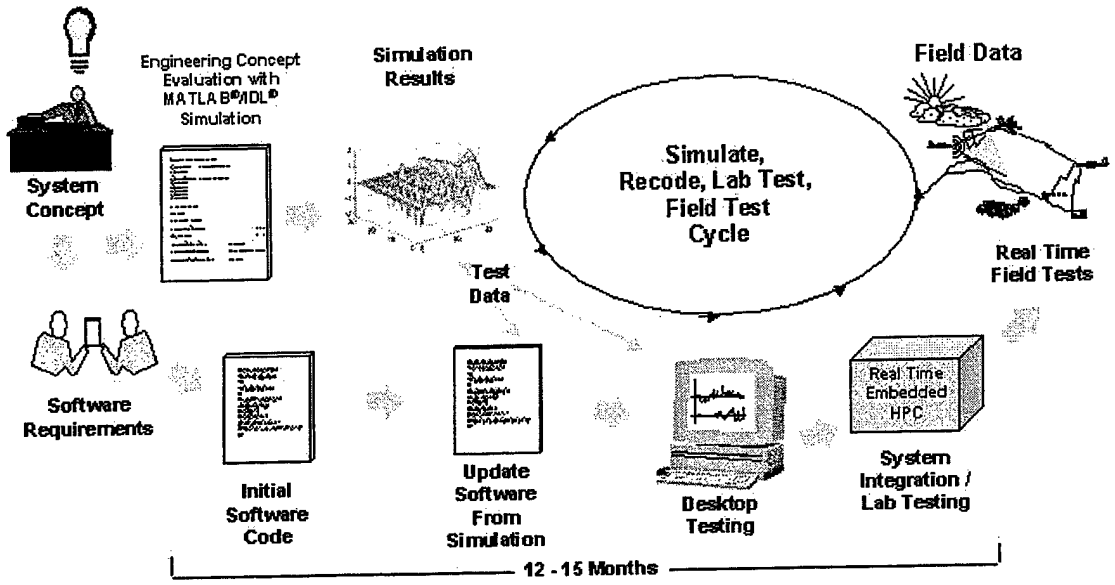


Figure 1 Typical Software Development Process Cycle

The approach presented in figure 1 requires multiple steps to be performed in order to move the adaptive signal processing application from concept to implementation. The systems engineer designs the adaptive signal processing algorithm in MATLAB®. The correctness of the algorithm is verified by running simulations. Once the correctness of the MATLAB® algorithm implementation has been verified, the algorithm description is handed off to the software engineer who then rewrites the algorithm in the appropriate programming language for the target signal processing hardware. Typically the

MATLAB® is rewritten in the C programming language. The C implemented version of the algorithm must then be compared to the MATLAB® implementation to insure that it provides the same functionality. Once it has been verified that the functionality matches, the C implemented algorithm can be executed on the target signal processing hardware. The results of the execution on the target signal processing hardware must then be compared with the results obtained from running the MATLAB® simulations to verify correctness. If inaccuracies are detected between the two versions, modifications must be made. This can result in multiple reiterations of the whole process. Constant coordination between the systems engineer and the software engineer must exist; if this coordination does not occur valuable information is lost. It is often the case that valuable information is lost.

ISI's innovative approach will reduce the typical software development process cycle. This modified software development process cycle is depicted in figure 2.

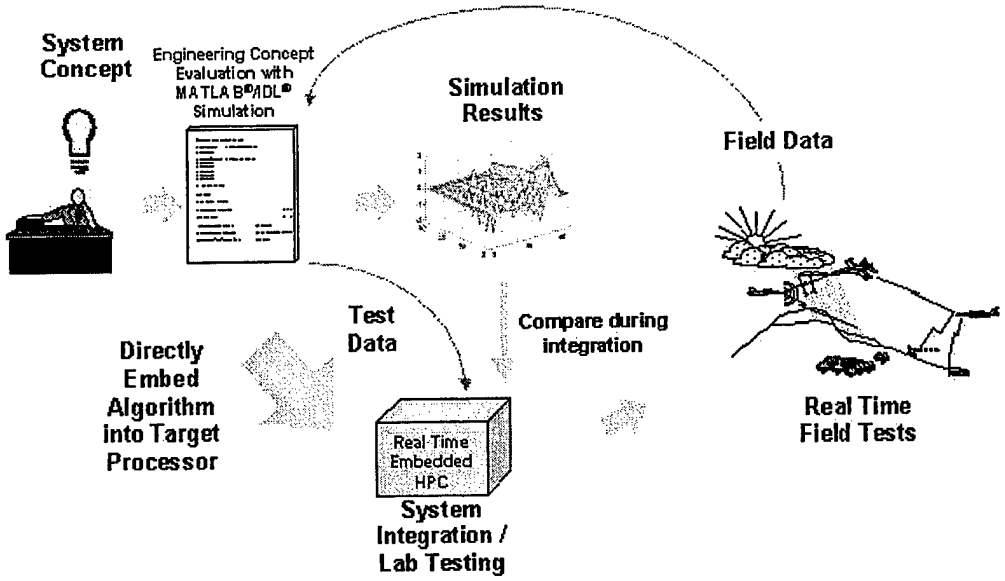


Figure 2 Modified Software Development Process Cycle

The original software development process cycle is significantly reduced. The systems engineer can still design and develop their adaptive signal processing algorithm in MATLAB®, and then test and verify its accuracy in MATLAB®. But the MATLAB® implementation no longer needs to be rewritten for the target signal processing hardware. The MATLAB® implementation can be executed directly on the target signal processing hardware. This capability is made possible by using MADS.

Figure 3 illustrates the MADS approach, where algorithm representation is based on a matrix based language well suited to radar signal processing. The MADS system depicted in figure 3 provides the capability for a user from their workstation platform, to develop radar signal processing algorithms using the MATLAB® language. The user can then simulate the performance of the algorithm in the MATLAB® environment. Once the user is satisfied with the simulated performance using MADS, the MATLAB® code can be

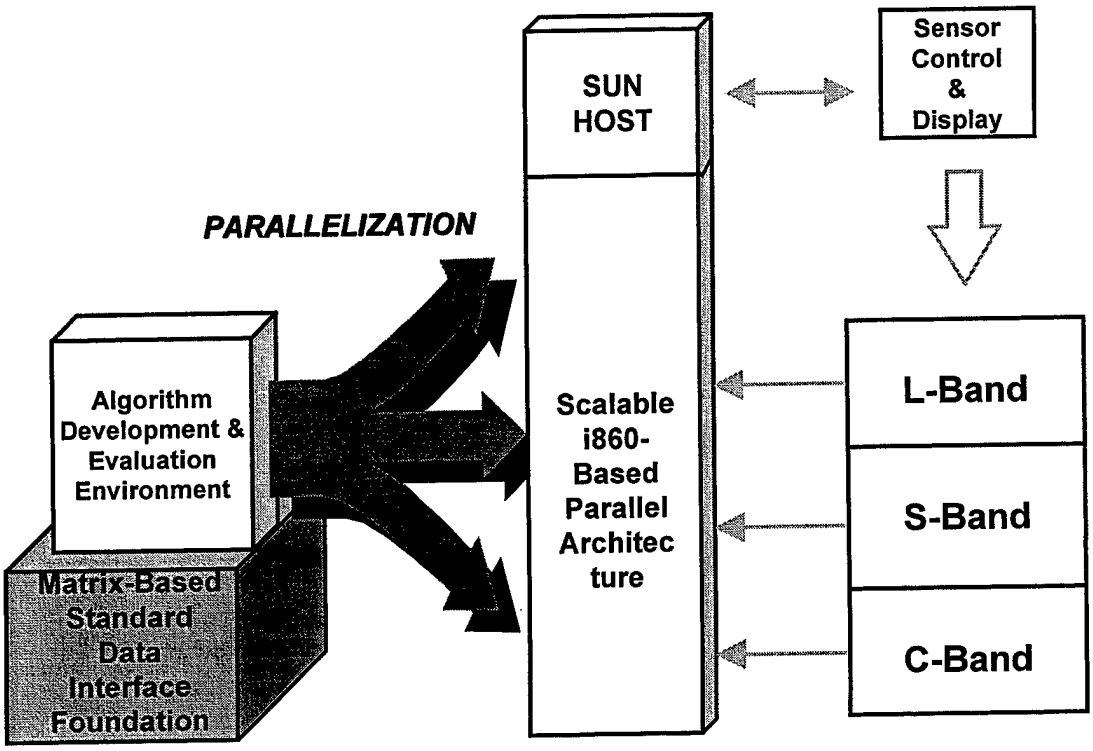


Figure 3 Multisensor Algorithm Development System (MADS)

mapped onto the i860-based vector processors. MADS will then generate the necessary C language code to run the application on the target i860-based vector processor architecture. MADS provides the capability to move from desktop development to a parallel implementation suitable for real-time multi-sensor experimentation quickly and efficiently.

This MADS concept has resulted in an open-ended test and evaluation environment, bypassing the lengthy hand-coding step for a new candidate algorithm. Integral to this environment is a graphical user interface designed to easily set up, control, and evaluate a variety of experiments. Figure 4 shows the conceptual organization of the MADS environment. The user communicates with the MADS environment using a SUN workstation terminal. MADS provides both a graphical user interface (GUI) and a command line interface (CLI). The GUI interface provides the capability to open both display and command line windows, and to build algorithms graphically by linking radar function icons. The GUI sits on top of the CLI interface, which translates graphical

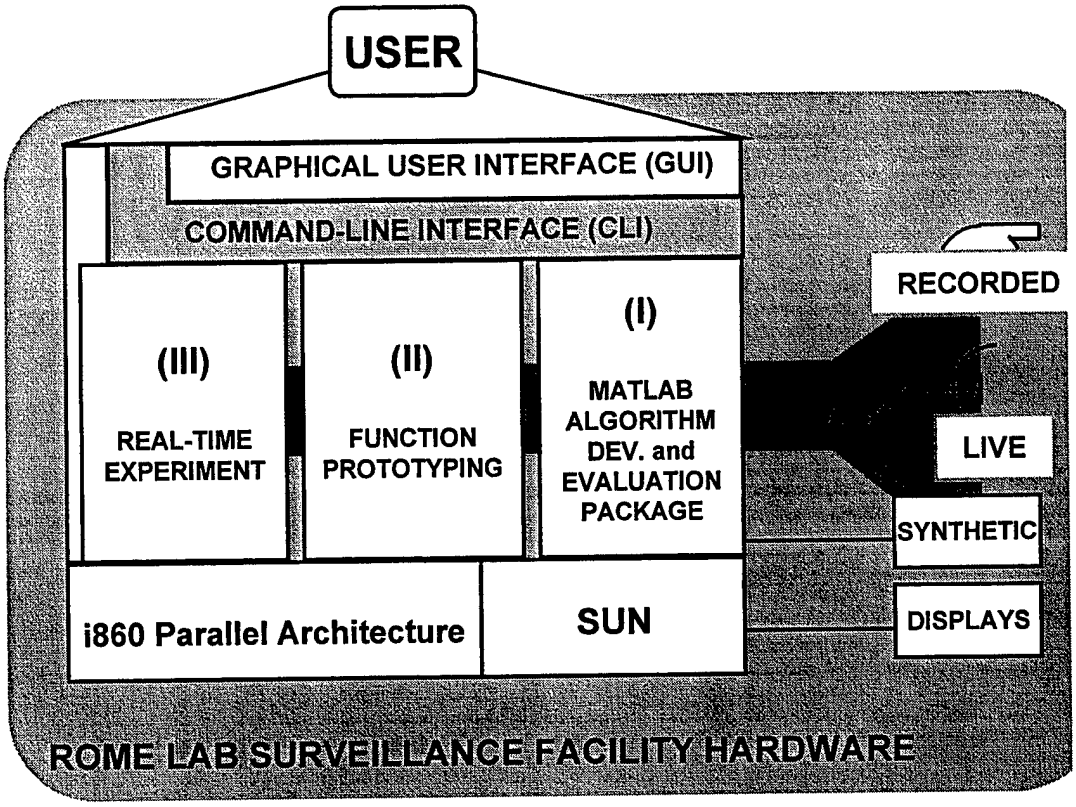


Figure 4 Conceptual Organization of the MADS facility

commands, stores and saves script files, and offers a quick access capability to a large function library. Below the CLI level are three different modes that the user can choose to work in. There is a non real-time algorithm development and evaluation mode. In this mode the user develops algorithm concepts in MATLAB® then tests the concepts using synthetic or stored data. The second mode is called the real-time function prototyping mode. In this mode the user takes a set of functions or subfunctions written in MATLAB® and maps them directly to execute on the i860 vector based architecture. The user is able to evaluate the throughput of portions of the algorithm, identify excessive processing loads, and modify or tweak the algorithm. The third and final mode is referred to as the real-time experiment mode. In this mode the user evaluates the full algorithm chain. The MADS environment provides the capability for live data to be read in, live data to be recorded, or synthetic data to be created.

The MADS hardware architecture testbed consisted of a SKY i860-based vector processor and a SUN computer to support real time displays.

In order to validate and verify the functionality of MADS, various classical radar signal processing functions were implemented. These radar signal processing functions were later used to develop a multi-sensor real-time detection and tracking experiment utilizing the AFRL S-Band and L-Band radars. These were demonstrated in real-time using the RSL facility.

2.2 MADS Environment

Figure 5 depicts the components of the MADS environment.

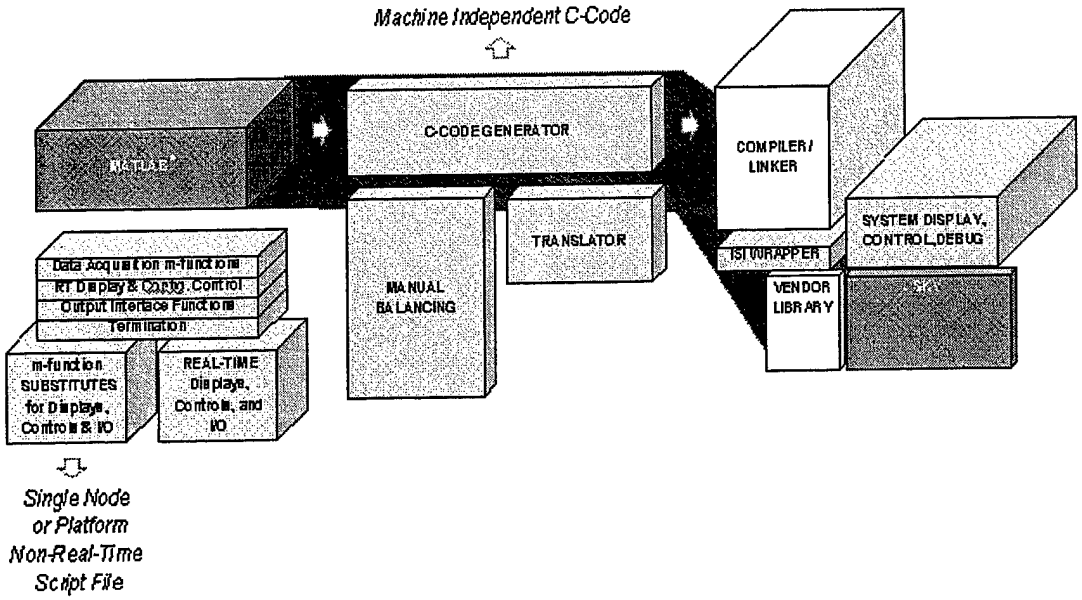


Figure 5 MADS Environment Components

The input to the MADS environment is a MATLAB® adaptive signal processing algorithm. Before using MADS, a user must verify that their algorithm runs correctly and error-free in the MATLAB® environment. The user can then use the MADS environment manual balancing tool to specify how their MATLAB® implementation will be partitioned to execute on the target signal processing hardware. The initial implementation of MADS supported the SKY i860 vector based processing architecture. Once the user has specified the partitioning, MADS then translates the MATLAB® code to C code. The MADS environment contains a translator based on lexical analysis and code generation that parses MATLAB® input code and generates a highly function oriented C-program. The translator is based on the GNU tools "flex" and "bison", which are the general equivalent of the UNIX tools lex and yacc. The translator supports all MATLAB® constructs including implicit looping denoted by the MATLAB® colon ":" operator. Complex concepts such as indexing matrices with vectors are also supported. Actual MATLAB® operations for both computation and data organization are implemented through the use of libraries developed as a part of ASPAE effort. The libraries do not support the notion of distributed matrices, since parallelism in MADS is realized by the mechanism of round robin data shuffling. A round robin parallelization scheme is typically used when the rate of data coming into the system is faster than the rate a subset of the processors can process the data. Parallelization is obtained by the fact that multiple input data sets are being processed in a staggered fashion simultaneously. The C code generated by the MADS translator is then compiled and linked for the target signal processing hardware using the vendors C compiler. It is at this point that the MADS function library, and the vendor specific libraries are linked in. The MADS

function library contains implementations of many of the MATLAB® functions. The vendor specific libraries contain hand-coded, machine specific vector functions that MADS uses to increase performance. The final step is to execute the C code on the target signal processing hardware. MADS provides the user with data visualization and control of their program. These visualization capabilities help assist the user in determining how well their algorithm is performing on the target signal processing hardware.

Figure 6 is a block diagram of a typical signal processing system configuration

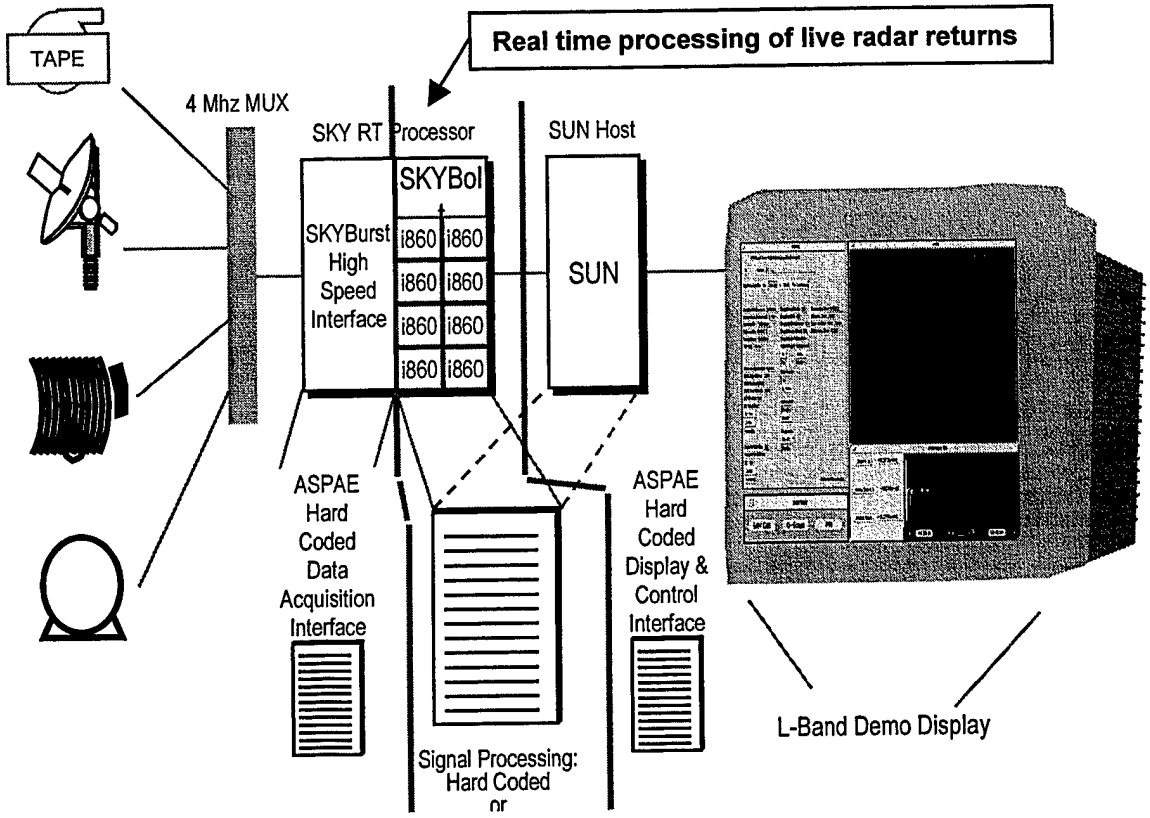


Figure 6 System Configuration of Typical Experiment

experiment in MADS. Data from one or more sensors or recorded data enters the SKYBurst interface through a MUX. Data routing software routes the data to a particular processor in the SKY multiprocessor. Routing is controlled through a hard-coded mechanism and directed by control information specified through the user interface.

Once on a particular i860, processing is controlled by the translated MATLAB® algorithm script in real-time. The parallelization paradigm used in this example is a round-robin scheme.

Processing results are reported to the user interface via the VME bus and MADS display. Figure 7 presents a collection of the types of control and data visualization objects supported by MADS.

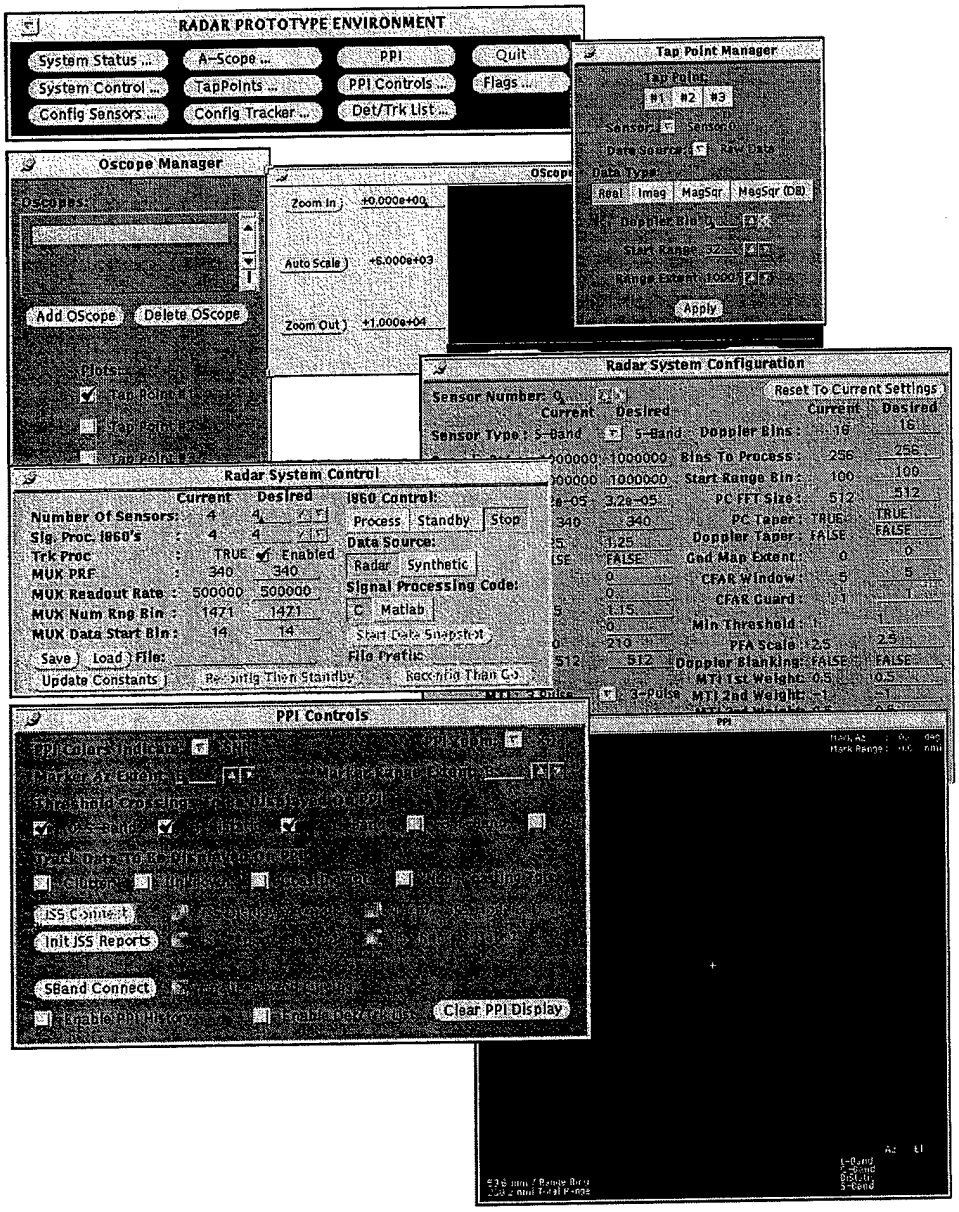


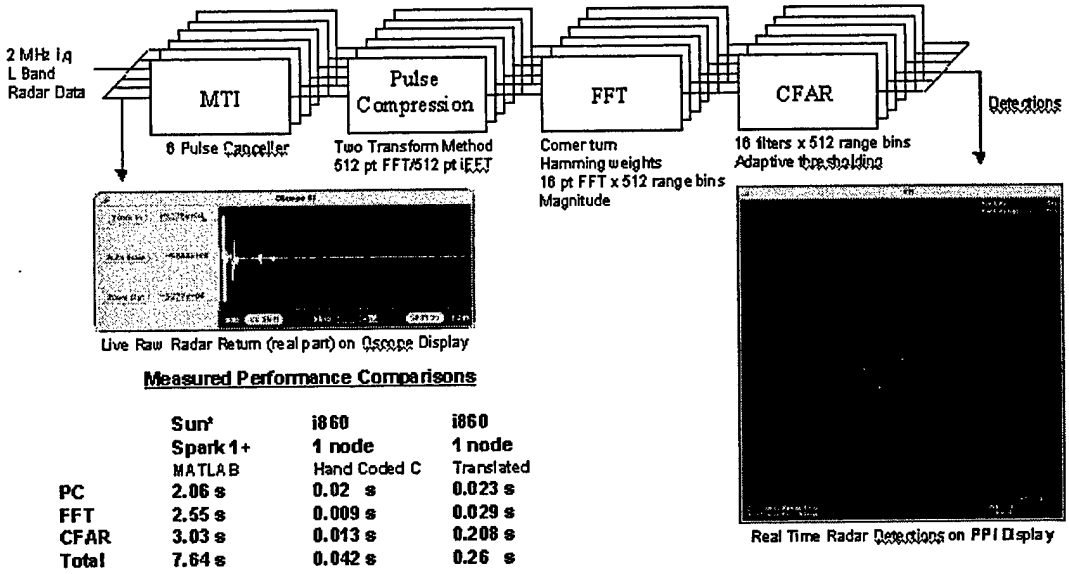
Figure 7 Control and Data Visualization Objects

The MADS visualization interface includes process control, status monitoring, configuration windows, tap point selection, real-time displays, and textual detection lists. A multitude of radar input and processing parameters can be set so that processing power can be brought to bear on the data segments of interest. Tap-points can be used to select predefined signal processing data objects to instrument for visualization. O-scopes can be

created and mapped to tap point data for a real-time oscilloscope representation of data. In a similar manner a Plan Position Indicator (PPI) display can be utilized to visualize detection and track data. The PPI display contains grey scale for aging detections, azimuth indicator dot, and azimuth range read out. For a truth reference and to identify targets of opportunity, live JSS feeds are supported.

2.3 Radar Signal Processing Application Developed Using MADS

Figure 8 presents a radar signal processing application that was developed using MADS and MATLAB®. By using this approach the application development time was



Measured Performance Comparisons

	Sun* Spark1+ MATLAB	i860 1 node Hand Coded C	i860 1 node Translated
PC	2.06 s	0.02 s	0.023 s
FFT	2.55 s	0.009 s	0.029 s
CFAR	3.03 s	0.013 s	0.208 s
Total	7.64 s	0.042 s	0.26 s

Figure 8 Radar Signal Processing Application

significantly reduced due to the fact that once the application was developed using MATLAB® it could be directly executed on the target signal processing hardware. The time saved is a result of not having to rewrite the MATLAB® implementation in another language.

In this application, radar signal processing application consists of four functions. The functions include Moving Target Indicator (MTI), Pulse Compression, Fast Fourier Transform (FFT), and Continuous False Alarm Rate (CFAR). The MTI function is a six pulse digital filter designed to suppress the mainlobe clutter. The pulse compression function performs the matched filtering necessary to compress a long pulse to a duration that is inversely proportional to the bandwidth. The FFT function performs the Doppler processing allowing moving target discrimination. The CFAR processor function adjusts detection threshold levels as a function of the local range-Doppler cell energy content to preserve the false alarm rate at some predetermined value. The input data is raw I and Q, L band sensor data. This data was read in from an L band sensor at AFRL as it was functioning. Figure 8 displays the results of multiple executions of the radar signal processing application. Three different results are provided for comparison. The three

different results are for the following: the radar signal processing application written in MATLAB® and executed in the MATLAB® environment, a hand-coded C implementation, and the MATLAB® implementation translated using MADS. The initial tests show that performance efficiencies of 40% to 87% of hand coded C can be obtained using MADS depending upon the function. The dataflow process through the MADS system used for building the radar signal processing application is depicted in figure 10.

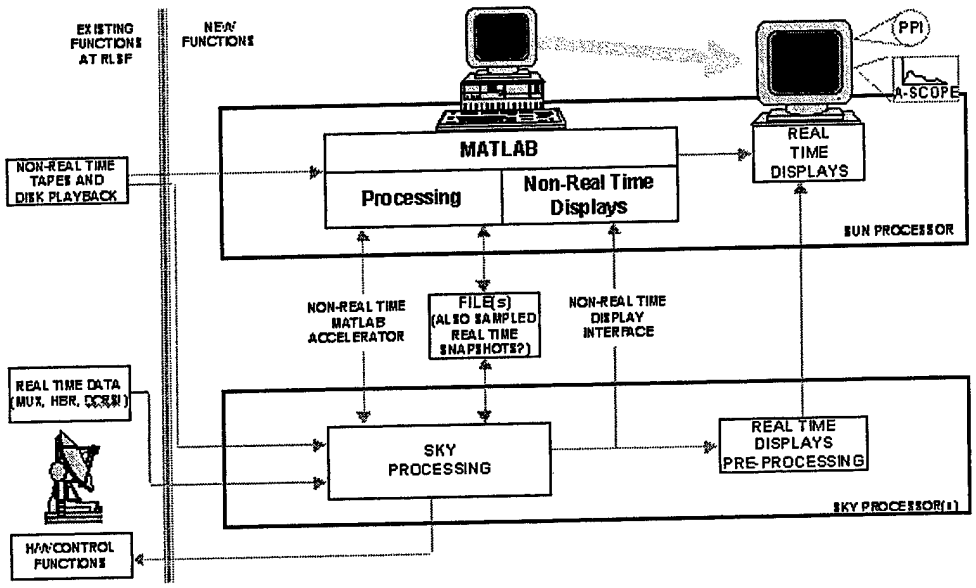


Figure 9 Dataflow Through MADS

The MATLAB® rapid prototyping concept innovated in MADS under the ASPAE program proved so effective that follow-on DARPA sponsorship was obtained for further research. The following section describes the evolution of the MADS concept.

3 Evolution of MADS

The MADS environment under the ASPAE program formed the foundation for the follow-on DARPA/ITO funded program entitled "Real-Time Development Environment (RTDE)".

3.1 Real-Time Development Environment (RTDE) Concept

The objective of the RTDE program was to develop a rapid prototyping environment that would provide a complete facility for a systems engineers to take algorithms written in MATLAB® and automatically generate executable parallel code suitable for embedded real-time execution and data analysis on parallel high performance computers. The RTDE significantly extended the existing MADS environment. The extensions include the following:

1. A parallel MATLAB® function library

2. Support for multiple parallelization paradigms including round robin, task parallel, data parallel, pipelined
3. Automatic generation of parallel C code
4. Execution on multiple real-time embedded platforms
5. Transparent automatic data distribution
6. Real-time performance visualization.

The RTDE incorporated five key features. These five key features are a translator for parallel code generation, standard message passing library, real-time support, parallel algorithm decomposition, and performance visualization. The translator converts MATLAB® source to C code and features the use of standard optimized mathematical library calls. A standard message-passing library is utilized to support the distributed aspect of a scalable implementation. Support is included to address real-time issues such as interrupts, data buffering and queuing, throughput, bandwidth, and latency. Semantics are provided for dealing with I/O data streams over standard interfaces. Parallel implementation of the overall algorithm is supported through user specified domain decomposition, functional decomposition, or a hybrid solution, as well as parallel libraries for functions within the application. Performance visualization is included to support test, integration, and evaluation.

The RTDE provides a toolbox of machine-independent, real-time utilities that the user can embed in a MATLAB® file to configure data flow, display, and control. A graphical Target Balancing Tool (tbt) allows the user to perform load balancing with manual allocation of MATLAB® processes to physical hardware nodes. The MATLAB® file is then translated to parallelize the functions to produce parallel C code that is machine-independent. Once the parallel C code is produced, the user can compile and link it for any of the supported target platforms. RTDE provides wrappers around third party libraries, which map script file function calls to the target library. The result of the compile and link stage is a set of real-time executable processes that acquire data from a continuous real-time data stream and distribute it across the platform's nodes, process the MATLAB® algorithm, and combine and output continuous data in real time. During execution, the user can monitor real-time displays that are created by the real-time toolbox functions, reconfigure user selected parameters, and observe throughput and loading performance. The RTDE functional tools are fully integrated to provide a seamless environment for all of the development needs of an embedded real-time system.

3.2 RTDE Components

Figure 9 depicts the physical components that comprise RTDE. RTDE consists of a workstation referred to as the HOST PC, which is utilized to interface with the target high performance computer. The HOST PC is used to create, compile, and link the parallelized C-code generated by RTDE. Like the MADS environment the input to the RTDE is a sequential MATLAB® m-file.

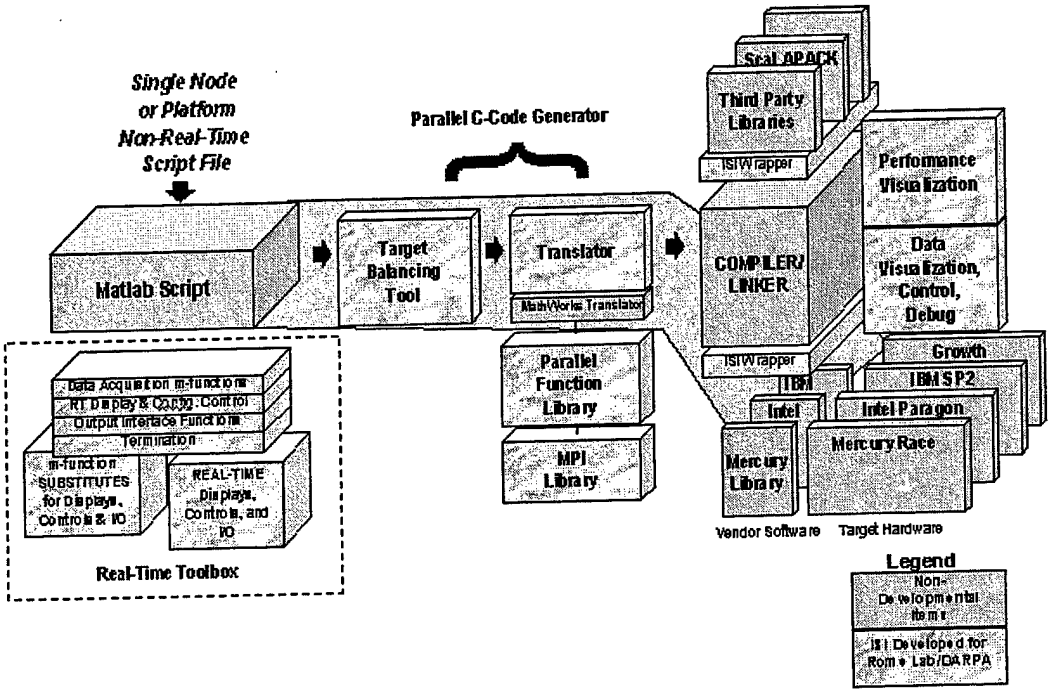


Figure 10 Real-Time Development Environment

The next component in RTDE is the target balancing tool (tbt). This component did not exist in the MADS environment. Tbt is used to modify the MATLAB® m-file, partition the m-file into groups and instances of groups, and specify the compile and run-time parameters for the selected embedded real-time high performance computer. A group in RTDE is defined as a set of tasks (comprised of MATLAB® scripts and functions) which are processed sequentially by each node of a user-specified set, or refers to the set of nodes dedicated to the tasks of the group. All the nodes within a group perform the same tasks in unison on different portions of the data. The output from the tbt is modified MATLAB® files. A sample of the tbt tool is depicted in figure 11.

The RTDE translator then processes the modified MATLAB® m-file. The translator that was used in MADS has been replaced by a three step translation process. The three-step translation process consists of: preprocessing, Mathworks Compiler, and post processing.

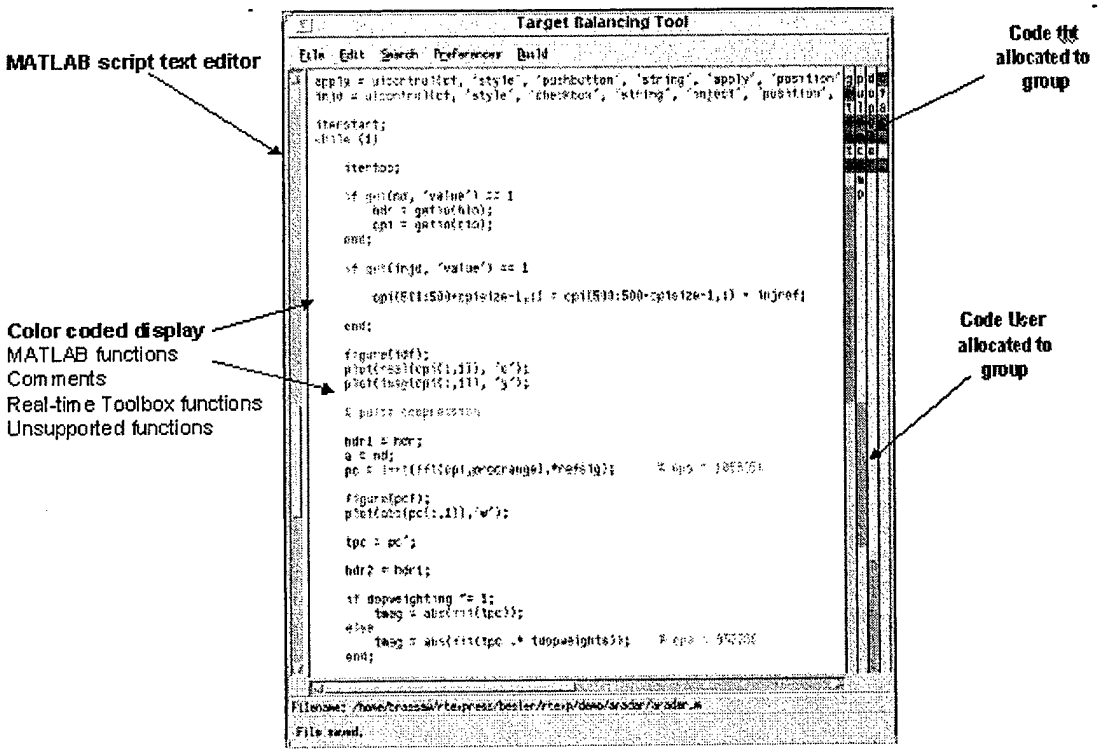


Figure 11 Target Balancing Tool

The preprocessing step is first performed on the m-file in order to prevent the Mathworks compiler from generating C code that will not parallelize well. Once the preprocessing of the m-file is complete, the m-file is translated to C code by the MATHWORKS compiler. This C code is then post processed. The post-processing step is required in order to incorporate support for single or double precision data types. MATLAB® only supports double precision data types, but RTDE supports both single and double precision data types. The user can specify whether they want single or double precision data using the tbt configuration control dialog.

Once the MATLAB® has been translated, the parallel C code is compiled and linked for the selected embedded real-time high performance computer. It is at this point that the RTDE parallel function library, message passing interface (MPI) library, third party libraries, and the vendor specific libraries are linked in. The RTDE parallel function library enhances and extends the function library that was developed in MADS by providing parallel implementations of the supported MATLAB® functions. For example, the parallel function library provides parallel implementations of functions such as FFT, and IFFT. The MPI library is linked in because RTDE uses MPI for providing communication between nodes on the real-time embedded parallel architectures. MPI was selected because it provides portability for RTDE across the supported parallel architectures. Third party libraries, for example ScaLAPACK, were utilized for the linear algebra parallel functions like eigenvalue, matrix inverse, QR reduction, and matrix

divide. The vendor specific libraries contain hand-coded machine specific vector functions that RTDE like MADS used to increase performance.

Once the parallel C code has been compiled and linked it can then be executed on the selected embedded real-time parallel architecture. RTDE Like MADS, RTDE provides data visualization/control of the application, but RTDE extends the real-time visualization capabilities of MADS by providing real-time performance monitoring. These visualization capabilities provide the user with feedback on how well their algorithm is performing on the selected hardware platform. RTDE added hardware architecture support for the Mercury Raceway i860 embedded real-time parallel architecture and the Intel Paragon i860 parallel architecture.

3.3 RTDE Real-Time Performance Monitoring

RTDE extended the MADS program execution visualization capabilities by providing a built-in display for parallelization performance statistics. Figure 12 provides a sample of the RTDE real-time performance monitor. The real-time performance monitor depicted in

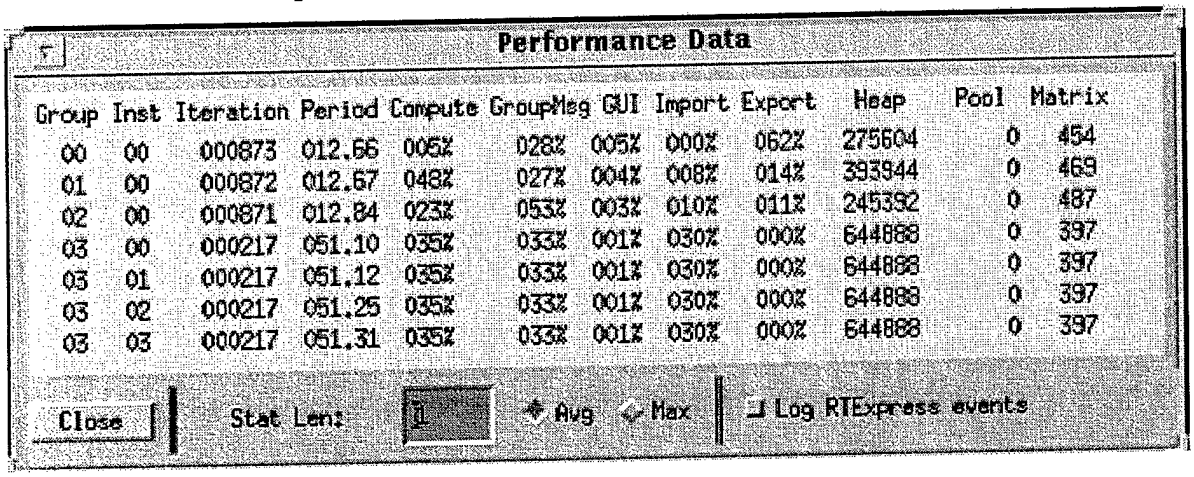


Figure 12 Real-Time Performance Monitoring

figure 11 provides information for each group (task) specified in the application program. In order to active the performance monitor for an application, the user must specify the portion of code that they wish to monitor. This is accomplished by using the real-time toolbox commands itertop and iterbot. The itertop command is placed above the first line of code where the monitoring is to begin. The iterbot command is placed after the last line of code to be monitored. For each group the following information is provided:

1. **Group:** The functional group number.
2. **Inst:** The instance number within the group (task).
3. **Iteration:** The number of times a designated code segment has been executed.
4. **Period:** The average CPU time spent within a designated code segment.
5. **Compute:** The percentage of time spent to compute the last iteration.
6. **GroupMsg:** The percentage of time spent on communication within the functional group.

7. **GUI:** The percentage of time spent of GUI data.
8. **Import:** The percentage of time spent getting/waiting to receive data.
9. **Export:** The percentage of tie spent sending/waiting to send data.
10. **Heap:** The size of the heap at the end of a designated code segment.
11. **Pool:** The number of buffers being used from a user defined static buffer pool.
12. **Matrix:** The number of active matrices at the end of a designated code segment.

3.4 RTDE Enhances MADS Capabilities

The DARPA/ITO funded RTDE program significantly extended the capabilities of the MADS environment developed under ASPAE. Table 1 presents a detailed description of the differences between the MADS environment and RTDE.

Table 1 - Comparison of Program Capabilities

Capability	Program	
	MADS	RTDE
General Goal	Provides a rapid algorithm prototyping facility based on MATLAB® and optimized for advanced radar signal processing on a Sky processor with a specific interface.	Provides a portable streamlined facility for efficient direct implementation of MATLAB® signal and image processing algorithms on real-time embedded parallel computing systems.
Parallelization Paradigm(s)	Round-Robin	Round-Robin Data Parallel Task Parallel Pipelined Combinations of Methods
Processor Target	Sky i860 Shamrock	Intel Paragon Mercury i860 RACEWay
Underlying Messaging	Native Sky API	MPI
Portability	Very limited	General
Application	Interactive	Interactive

Steering	checkboxes, text edit boxes, and buttons.	checkboxes, text edit boxes, and buttons.
Data Type Support	float (32-bit)	float (32-bit) or double(64-bit) but not both simultaneously
MATLAB®/C Translation	In-house designed translator utilizing Flex and Bison	In-house designed translator utilizing Flex and Bison, Mathworks MATLAB®-to-C translator, C-based post processor
MATLAB® Capability	Tailored specifically to radar signal processing	MADS capability extended to support a broader range of signal and image processing capabilities, including matrix factorization and inversion based on third-party libraries
Real-Time Constraints	Data and throughput driven	Data and throughput driven
Displays	Single plot, gray scale image, PPI, fixed size output.	Single plot, gray scale image, PPI, fixed size output.
Data Manipulation	Single node general Matlab 4.x matrix manipulation capability	Data parallel general Matlab 4.x matrix manipulation capability
Input/Output	SKYBurst Input	RINT, HIPPI
Software Testing	Functional testing based on test application end-results.	Individual functional testing with regression capability. Higher level testing based on demo suite.
Vector Acceleration	Sky i860 library	Paragon Kuck and Associates Accelerated library Mercury SAL
Performance Visualization	None	Group/Instance summary text display

Visualization		summary text display at run-time
Graphical support for user specified coarse grained parallelism	None	Target Balancing Tool (TBT)

Table 1 - Comparison of Program Capabilities

4 RTExpress™ Commercialization of RTDE

At the conclusion of the DARPA/ITO funded RTDE effort in 1997, ISI launched an internal development effort to provide a commercial implementation of the RTDE environment. The commercial implementation of RTDE referred to as RTExpress™ officially began selling in July 1998. RTExpress™ currently has many government, commercial, and university customers, and its user list continues to grow.

RTExpress™ also continues to grow as a product by adding new features and capabilities. In the architecture area, the RTExpress™ environment currently supports a wide variety of both embedded real-time and non-embedded real-time parallel architectures. The current supported architectures include: Mercury Raceway (PowerPC), CSPI (PowerPC), Intel Paragon, IBM SP2, Sun Enterprise, Network of Sun Workstations, Network of Linux PC's. In the future the SKY and SGI Origin 2000 architectures will be added.

RTExpress™ also includes additional visualization capabilities. One such new capability is a post-mortem instrumentation tool. This tool allows a user to instrument the execution of their MATLAB® program on the selected target architecture. It provides the user with a detailed presentation of their program execution. Figure 13 provides an example of the post-mortem visualization capability provided in RTExpress™.

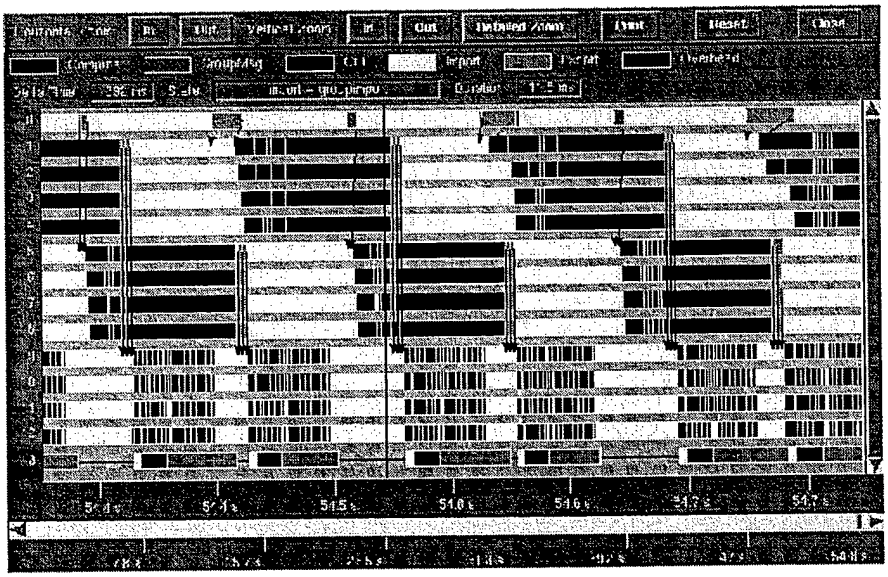


Figure 13 RTEshot Post-Mortem Instrumentation

5 Future Directions

The initial MADS environment concept will continue to grow both commercially as RTExpress™ and from a research direction under current DARPA/ITO funding. Under DARPA/ITO funding, research is currently being conducted to provide a Heterogeneous Real-Time Development Environment (H-RTExpress).

H-RTExpress will provide enabling technology to quickly and effectively move systems targeted for heterogeneous architectures from concept to implementation. An important part of such a tool is the optimization of the hardware selected to meet specific algorithm requirements, and optimal/portable, mapping of the algorithm onto that hardware. The tool suite will address the rapid changes in hardware which tend to quickly obsolete systems that do not include portable software, and thus lead to large software reinvestments.

The objective of H-RTExpress is to develop an integrated software development environment for heterogeneous systems. H-RTExpress will provide the following features:

- An integrated graphical user interface (GUI) environment for end-to-end application development on heterogeneous processor architectures in the system engineering domain.
- Techniques and notation for specifying variable data precision in conjunction with MATLAB® algorithm description.
- Methods for mapping MATLAB® code segments onto heterogeneous processing nodes consisting of both embedded and adaptive computing hardware (Field Programmable Gate Array Technology).
- Methods for launching applications across heterogeneous processor architectures

6 References

[1] MATLAB® Reference Guide. The Mathworks, Inc., Natick, MA, 1992.

[2] W Gropp, E. Lusk, A. Skjellum. Using MPI: Portable Parallel Programming with the Message Passing Interface. The MIT Press, Cambridge, MA 1994.

[3] M. Benincasa, R. Besler, D. Brassaw, R. Kohler. Rapid Development of Real-Time Systems Using RTExpress™. Proceedings of the 1998 IPPS/SPDP Symposium. pp. 594-599, April 1998.

[4] The RTExpress™ User's Guide. Integrated Sensors, Inc., Utica, NY,. 1998.

[5] M. Benincasa, R. Besler, D. Brassaw, J. Steill. RTExpress™ A Rapid Prototyping Environment for Real-Time Embedded Systems. Proceedings of the MIT Lincoln Laboratory HPEC Conference, Sept. 1998.